An Implementation of a Commercial Messaging System Standard for a Space Mission Application

Carlos Carrion
Mission and Systems Architecture Section
Jet Propulsion Laboratory
Pasadena, California 91109
carlos.carrion@jpl.nasa.gov

Abstract

Standardization of interfaces between on board spacecraft subsystems, operation control centers, and ground terminals is seen as a way to reduce overall mission development, integration and operations costs. Messaging systems used in the electrical utility, petrochemical, and automotive industries look promising for application to space based systems for command, control, and communications. These messaging systems provide a baseline set of capabilities useful for space based systems and mature protocol standards that have been shown to be applicable to space missions.

This paper describes work that is being done to use a currently available commercial standard messaging system to command, control, and communicate with several distributed systems generally used in unmanned space missions. These include a simulated ground terminal, a simulated control center, and a set of simulated spacecraft subsystems/devices. The messaging system services are implemented for each device in software modules that are referred to as 'virtual devices' containing the externally visible attributes (those that can be controlled or monitored) of the "real" device.

A rudimentary imaging mosaic mission scenario was implemented with less than 25% of the full messaging system services provided. As a result of this implementation, modifications to the messaging services used may be needed specially for deep space mission applications.

Introduction

Information interchange between on board spacecraft subsystems is one of the key problems in a space mission that is given high priority. A great deal of effort and resources are spent in developing a solution to this problem. This solution must be developed despite constraints placed on spacecraft development that are not typically put on ground system implementations. Such constraints usually are things like mass, volume, radiation hardened technology, and power. More often than not the solution is project specific and reusability or interoperability of a mission's data system and data interfaces is not a prime concern, at least in the deep space mission world.

In the commercial world where increasing profits and getting a return on investment are of prime concern, ways in which processing and manufacturing costs can be lowered are usually in demand. Thus in the early 1980's a group of device vendors began to propose draft standards for the transfer of digital information in the manufacturing environment. This work led to the development of ISO (International Standards Organization) standard 9506, the Manufacturing Message Specification (MMS)[1]. MMS and similar messaging systems are used in the automotive, petrochemical, manufacturing, and electric utility industries.

The application of commercial messaging system standards is not new to the space mission environment. Work has been done to apply MMS to the Space Station Freedom,[2] and a prototype monitor and control MMS based system for the Jet Propulsion Laboratory's (JPL) Deep Space Station 13 was done several years ago[3] which is still operational.

Applicability of Messaging Systems

One of the reasons messaging systems were developed was to lower the cost of integrating smart devices built by different vendors into a factory. A factory floor can be looked at as a set of distributed heterogeneous intelligent devices that need to communicate with each other. If one views a spacecraft as a set of interconnected devices that need to communicate with each other then there is inherently little difference between the two and therefore the use of messaging systems in space missions makes sense.

Since work had been done at JPL using MMS and expertise was available, it was chosen as the messaging system for this task. No other reason other than its availability was used to select MMS and the vendor which supplied the software.

Manufacturing Message Specification

In the OSI (Open Systems Interconnection) network communications model, MMS lies in the application layer (see Figure 1). This means that MMS does not provide connectivity functions (e.g., data routing and packet retransmission). Those functions are left to the layers below. MMS as an application layer protocol provides definition, structure, and meaning to messages being passed between devices. Applications like reasoning engines and analysis software are placed on "top" of a messaging system like MMS.

MMS is built upon a model called the Virtual Manufacturing Device (VMD). The VMD (see Figure 2) specifies how MMS devices (called servers) behave as viewed from an external MMS client application. This model defines objects (variables, events, and semaphores for example) that are contained in a server, the services that are used to manipulate objects, and the behavior of the server when requests are received from a client. In short the VMD is a representation of the network visible attributes (both controllable and monitorable) of the "real" device. A vendor of an MMS compatible device would deliver to the customer a package shown by the dashed outline box in Figure 2.

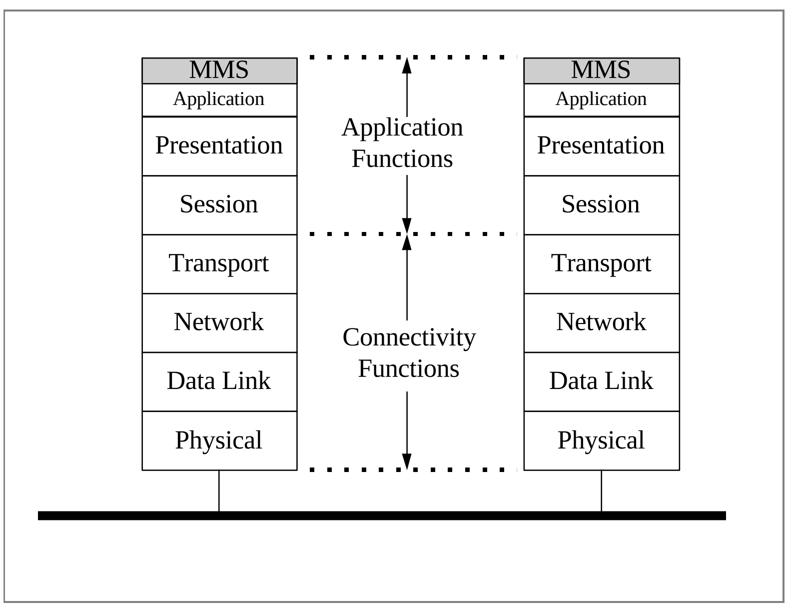

Figure 1 MMS in the OSI Network Communications Model

When a client issues a request (see Figure 2), the server receives the request (indication), does whatever it is programmed to do upon the receipt of the indication, generates an “answer” (response) to the client’s request, and the client receives the “answer” (confirmation) from the server. MMS provides for unsolicited responses from the server as well. This means that in certain cases a server doesn’t have to receive a request from a client to take action and issue a message.

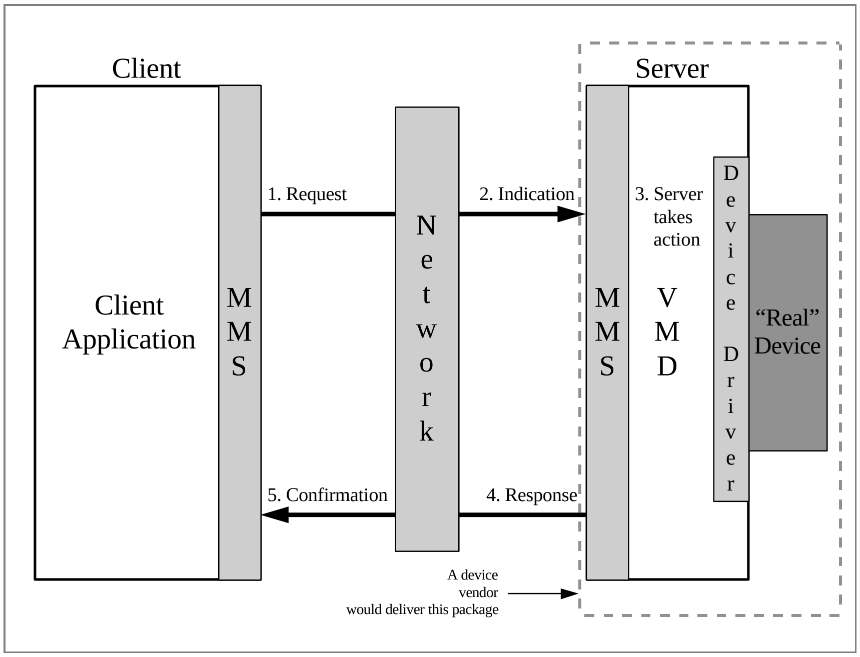

Figure 2 VMD Model & Client/Server Relationship

MMS provides 86 services ranging from initiating connections, to writing variables at a server, to creating token semaphores, to managing shared resources (see Table 1).

Table 1 MMS Service Categories and Functions

Categories of MMS Services	Functions Performed
Context Management Services	Used for initiating and concluding connections between a client and a server
VMD Support Services	Used for getting information about a server
Variable Access and Management Services	Used for reading and writing variables at a server
File Access and Management Services	Used for reading, obtaining, and getting information about files at a server
Event and Alarm Management Services	Used to define, enroll, and get information about events at a server
Domain Management Services	Used for uploading, downloading and getting information about domains at a server. Domains are regions of server memory that populate Program Invocations.
Program Invocation Management Services	Used for starting, stopping, and getting information about Program Invocations at a

	server. Program Invocations are regions of server memory that are executable. A Program Invocation may contain one or more domains.
Semaphore Management Services	Used to control, and get information about token semaphores at a server. Semaphores are used to manage shared resources (printers, mass storage, etc.).
Operator Communication Services	Used to obtain inputs from and display output to a human operator at a server.
Journal Management Services	Used to read, write, and get information from a journal at a server.

Scenario

Refer to Figure 3 for this discussion. The scenario that was chosen was the monitor and control of a mosaic (a set of images of size *m* rows by *n* columns) centered on a target. We used the Flight System Testbed (FST) at JPL for our simulated spacecraft. We chose to use two spacecraft subsystems: the Attitude and Articulation Control Subsystem (AACS) and the Camera Simulation. The Low Earth Orbit Tracking Station workstation (LEO-T) was used for the ground terminal server. The Simulated Mission Operations Control Center (SMOCC) was used as the development and runtime environment.

The functional blocks in the scenario are: a client that is used by the user to run the scenario, a mosaic controller that controls the execution of the mosaic, a camera that takes each image in the mosaic, an attitude control subsystem that performs the spacecraft turns to point the camera, and a ground terminal that informs the client of its status. The ground terminal was not used to establish the client to spacecraft communications link.

To start the scenario, the user connects to the mosaic controller, camera simulator and ground terminal. The user defines the mosaic by specifying the number of rows, number of columns, amount of overlap between images in each direction, and the target quaternion. This is done by writing to variables in the mosaic controller. The user then sets the exposure time, and selects the imaging filters for the images by writing to variables in the camera simulator. The user can also direct the camera simulator to compress the “images”. The user also initializes the simulated spacecraft via a setup script which starts the dynamics simulation (world model), the spacecraft radio, the spacecraft’s computer flight software (including the attitude control software), and the ground data system.

The user then directs the mosaic controller to start the mosaic execution. The mosaic controller then connects to the attitude control subsystem and to the camera simulator. The mosaic controller builds the mosaic command from the variables that were written by the user and sends the mosaic command to attitude control. The attitude control subsystem takes care of the interface to the simulated spacecraft in the FST and the spacecraft begins the turns. From this point on the execution is controlled by the mosaic controller which instructs the camera simulator to take an image after each turn completion. After the execution is complete, the mosaic controller informs the client that the mosaic has been completed.

At this point, the user can do several things: for example, download the “images” from the camera simulator, delete the “images” from the camera simulator file store, instruct the camera simulator to execute one of two stored maintenance scripts which can be overwritten by the user via the client. The user then closes the connections to the mosaic controller, camera simulator, and ground terminal. The mosaic controller then closes its connections to the attitude control subsystem and camera simulator.

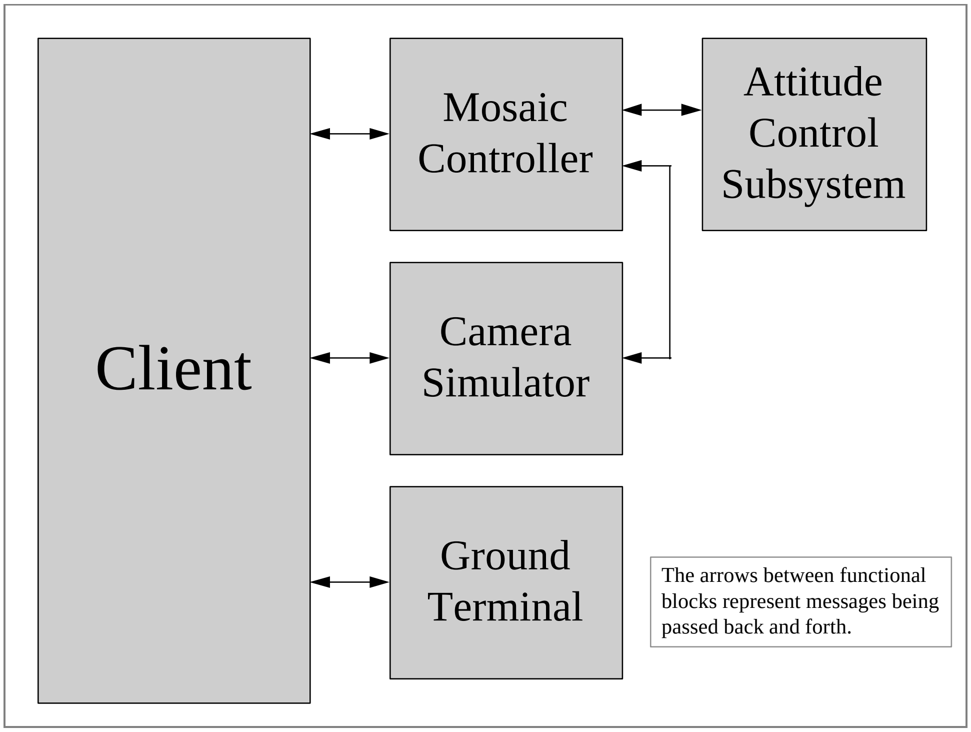

Figure 3 Mosaic Scenario Configuration

Implementation

An OSI network communications stack software package (Sun Microsystems’ SunLink OSI version 8.1) was installed in the SMOCC hosting the client, the mosaic controller, the attitude control subsystem, and camera simulator servers to provide the network communication layers needed by MMS. RFC-1006, provided by the stack package, was used as the network device to access the TCP/IP network at the transport layer (see Figure 1). This was done to use the TCP/IP network already in place at JPL and to use simple internet addresses for the client and the servers. The MMS implementation (MMS-EASE-133-015 V6.0.3) was procured from Systems Integration Specialists Company, Inc. All code was compiled and linked with standard Sun Solaris utilities.

Since the FST environment is not MMS based, modifications to the FST camera simulation code were done to make it MMS compatible. The basic functionality of the camera was kept, including image compression, but to keep this scenario simple, the camera simulator interface to the world model software was removed. Thus the images the camera simulator returns are strings of bytes. The modified camera simulator code was moved from the FST to the SMOCC and compiled.

The attitude control server that was built for this scenario acts as the interface to the spacecraft in the FST. The attitude control server receives the mosaic command from the mosaic controller, passes each individual turn command to the FST, and receives event notifications from the FST (e.g., turn begin and turn end) based on the computations done by the simulated spacecraft's flight software and the world model software. The attitude control server itself does no attitude computations.

The LEO-T MMS server was not built specifically for this scenario, but it was used as a passive device to demonstrate the applicability of MMS to monitor and control the subsystems at a ground terminal. The LEO-T server is not controllable at this time. Only monitor information is received from the server. The LEO-T was not used to establish any kind of client to spacecraft communications link.

18 user defined variables were used in the scenario (see Table 2). The 4 variables used in the ground terminal server to report monitor data to the client are data structures and as such were not part of the MMS-EASE standard data types (i.e., float, double, integer16, integer32, unsigned8, byte, ...). They had to be defined both in the client and ground terminal server. MMS-EASE provides for user defined data types (i.e., arrays, structures, nested structures, ...).

Table 2 User Defined Variables in the Mosaic Scenario

MMS Server	User Defined Variables
Mosaic Controller	nrows ncols x_overlap y_overlap center_quat start_mosaic
Attitude Control	aacs_cmd picture_taken
Camera Simulator	image_compression filter_wheel_pos_1 filter_wheel_pos_2 camera_exposure camera_status transmit_image
Ground Terminal	AntennaData SchedulingData FramingData PassInfoData

5 events were defined and enrolled for notification (see Table 3).

Table 3 Events Defined in the Mosaic Scenario

MMS Server	Events	Defined by	Notify
Mosaic Controller	MosaicBegin MosaicEnd	Client Client	Client Client
Attitude Control	TurnBegin TurnEnd	Mosaic Controller Mosaic Controller	Mosaic Controller Mosaic Controller
Camera Simulator	ShutterClosed	Client	Client Mosaic Controller

2 domains were created and defined in the camera simulator. 2 program invocations were created and defined in the camera simulator. Each program invocation contains one domain.

Table 4 Domains and Program Invocations Defined in the Mosaic Scenario

MMS Server	Domains	Program Invocations
Camera Simulator	script1 script2	run_script1 run_script2

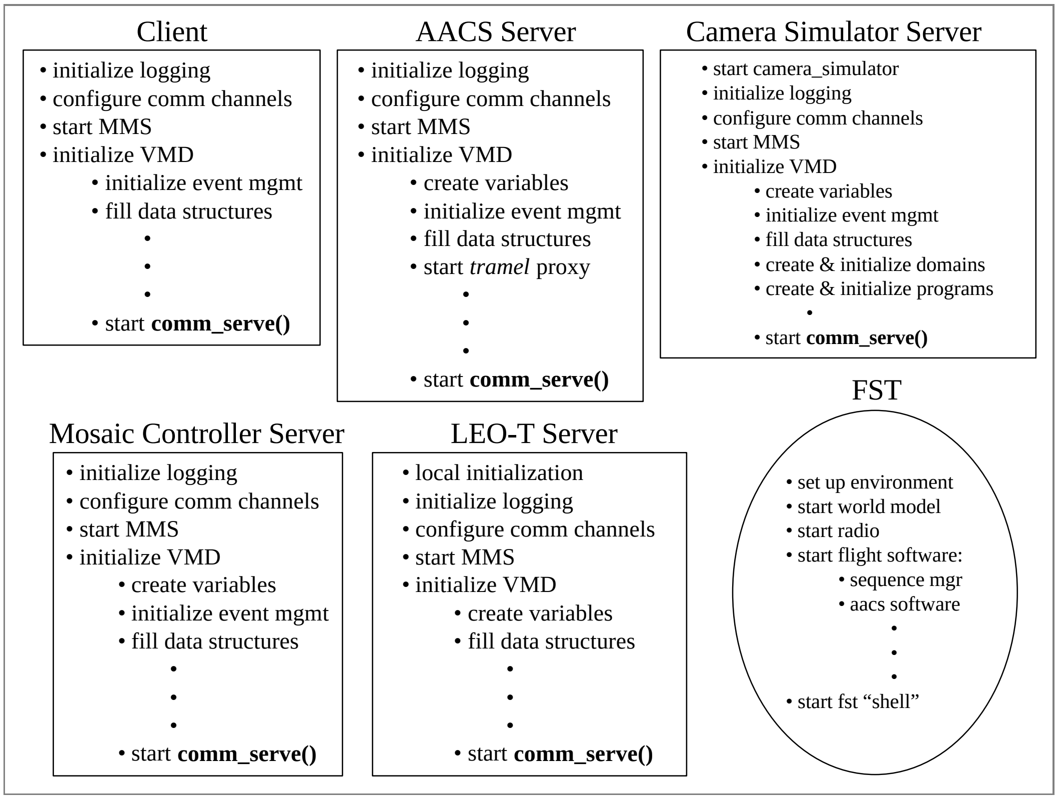

Figure 4 Initialization of Mosaic Scenario Client & Servers

Initialization of Client and Servers (see Figure 4):

1. The client and all the servers are started by a simple UNIX script. Upon execution each server and client initializes the MMS logging features, communication channels, and whatever local initialization needs to be done, e.g., the camera simulator and the LEO-T. Then MMS is started and each VMD is initialized (variables are created, events are defined, domains are created, program invocations are created, populated with domains, and set to the correct state, etc.).
2. The MMS-EASE event loop, ***comm_serve()***, is started for the client and each server.
3. The simulated spacecraft and its environment are started and initialized in the FST.

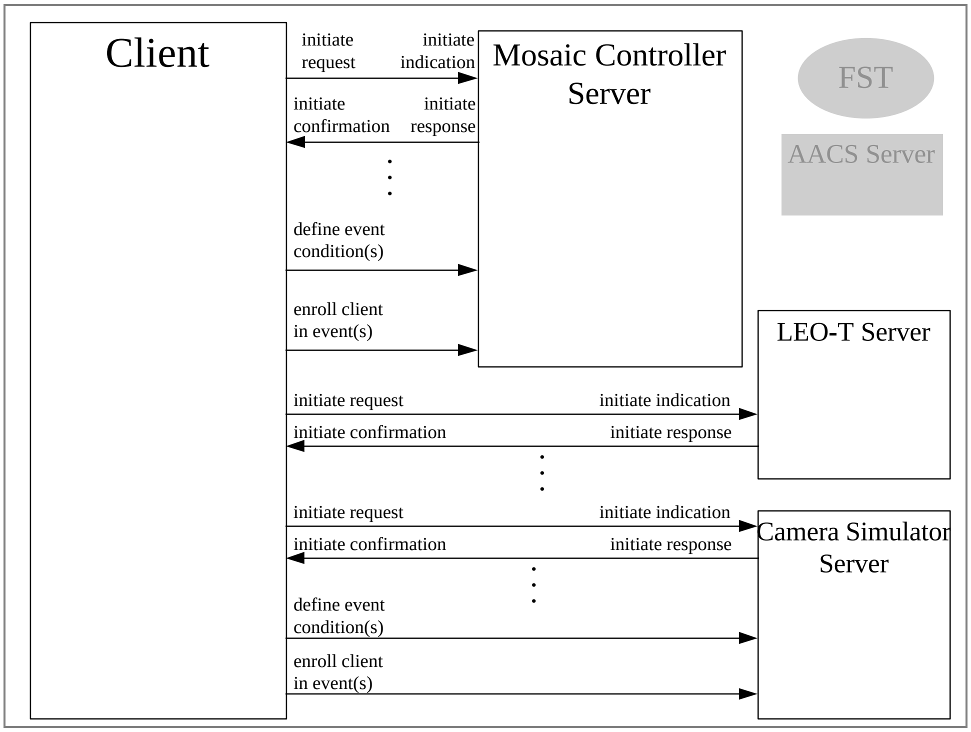

Figure 5 Initiation of Connections by the Client

<u>Initiation of connections with the spacecraft and ground terminal by the Client (see Figure 5):</u>

4. The Client initiates a connection with the Ground Terminal.
5. The Ground Terminal responds positively to the *initiate* message.
6. The Client requests the Ground Terminal to identify itself.
7. The Ground Terminal responds with identifying information. As soon as the connection is established, the Ground Terminal starts sending information reports to the Client.
8. The Client initiates a connection with the spacecraft's Mosaic Controller.
9. The Mosaic Controller responds positively to the *initiate* message.
10. The Client requests the Mosaic Controller to identify itself.
11. The Mosaic Controller responds with identifying information.
12. The Client sends *MosaicBegin* and *MosaicEnd* event definition requests to the Mosaic Controller. For each event definition there is an indication received by the Mosaic Controller, a response generated by the Mosaic Controller, and a confirmation received by the Client.
13. The Client sends requests notifying the Mosaic Controller that for each event, the Client would like to be notified when the event is triggered. For each request, a response is generated by the Mosaic Controller and a confirmation is received by the Client.
14. The Client initiates a connection with the spacecraft's Camera.
15. The Camera responds positively to the *initiate* message.
16. The Client requests the Camera to identify itself.
17. The Camera responds with identifying information.
18. The Client sends a *ShutterClosed* event definition request to the Camera.
19. The Client sends a request notifying the Camera that the Client would like to be notified when the event is triggered. A response is generated by the Mosaic Controller and a confirmation is received by the Client.

<u>Initiation of connections on-board the spacecraft (see Figure 6):</u>

20. Variables in the Camera are written by the Client: *image_compression*, *filter_wheel_pos_1*, *filter_wheel_pos_2*, and *camera_exposure*.
21. For each write variable request by the Client, there is an indication received by the Camera. The Camera generates a response, and the Client receives a confirmation of the write variable request.
22. Variables in the Mosaic Controller are written by the Client: *nrows*, *ncols*, *x_overlap*, *y_overlap*, and *center_quat*.
23. For each write variable request by the Client, there is an indication received by the Mosaic Controller. The Mosaic Controller generates a response, and the Client receives a confirmation of the write variable request.
24. The Client sends a request to the Mosaic Controller commanding the spacecraft to execute the *mxn* mosaic by writing to the variable *start_mosaic*.
25. The Mosaic Controller upon receipt of the request builds the mosaic command from the variables that were written by the Client.

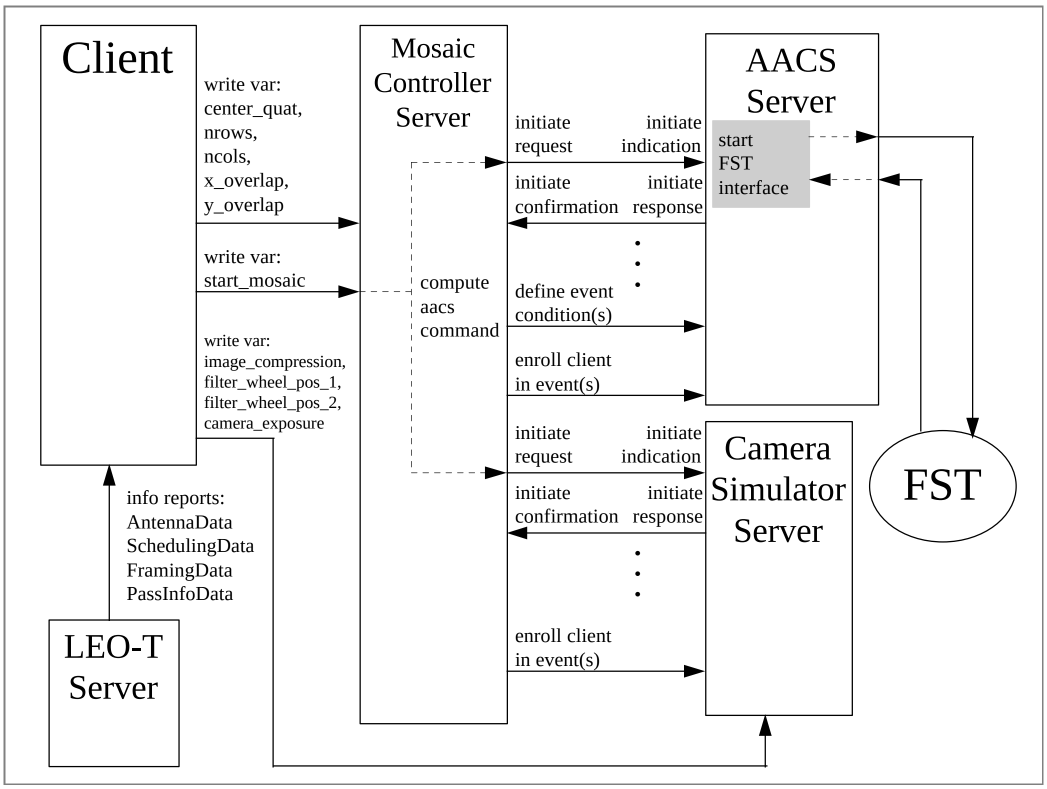

Figure 6 Initiation of Connections On Board the Spacecraft

26. The Mosaic Controller initiates a connection with the spacecraft's Camera.
27. The Camera responds positively to the *initiate* message.
28. The Mosaic Controller requests the Camera to identify itself.
29. The Camera responds with identifying information.
30. The Mosaic Controller sends a request notifying the Camera that it would like to be notified when the *ShutterClosed* event is triggered. A response is generated by the Camera and a confirmation is received by the Mosaic Controller.
31. The Mosaic Controller initiates a connection with the AACS.
32. The AACS responds positively to the initiate message.
33. The AACS establishes a connection to the FST.
34. The Mosaic Controller requests the AACS to identify itself.
35. The AACS responds with identifying information.

36. The Mosaic Controller sends *TurnBegin* and *TurnEnd* event definition requests to the AACS. For each event definition there is an indication received by the AACS, a response generated by the AACS, and a confirmation received by the Mosaic Controller.
37. The Mosaic Controller sends requests notifying the AACS that for each event, it would like to be notified when the event is triggered. For each request, a response is generated by the AACS and a confirmation is received by the Mosaic Controller.

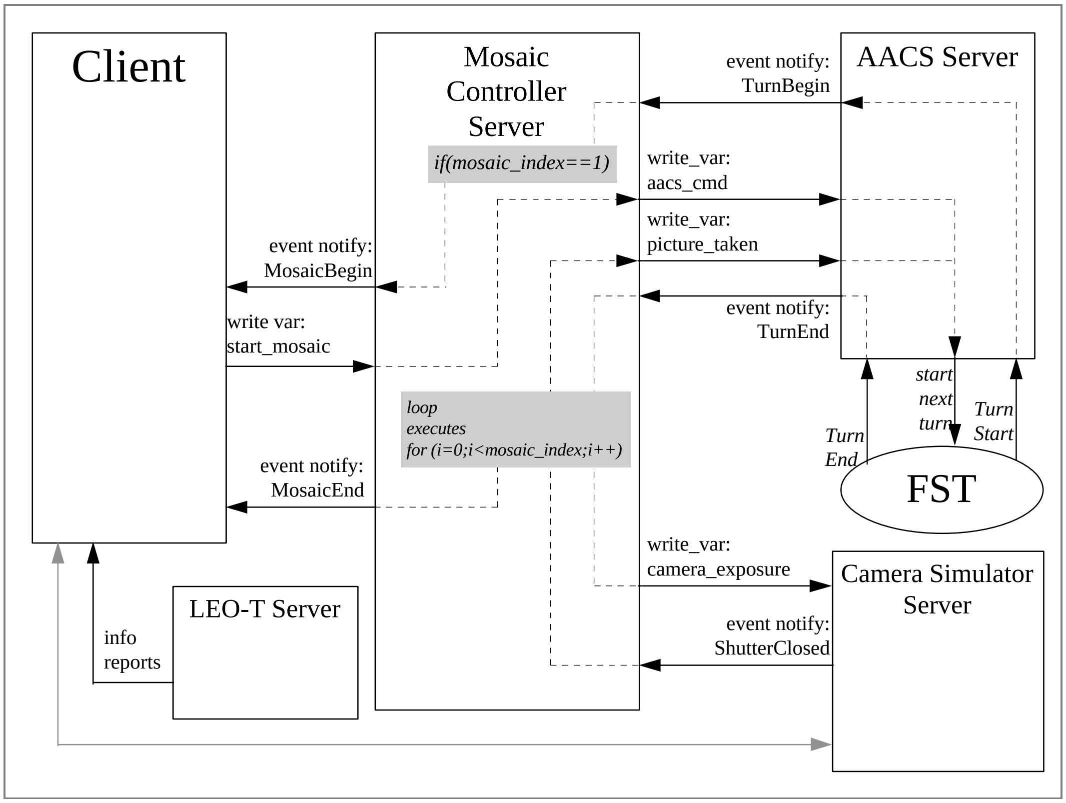

Figure 7 Mosaic Execution

<u>Execution of the mosaic command (see Figure 7):</u>

38. The AACS sends the first turn command to the FST.
39. The FST spacecraft and world model perform all the computations to turn to the desired attitude.
40. The FST informs the AACS that the turn to attitude has begun. The AACS issues a *TurnBegin* event notification to the Mosaic Controller. If it is the first turn then the Mosaic Controller issues a *MosaicBegin* event notification to the Client.
41. When the attitude is reached, the FST informs the AACS that the turn has been completed.
42. The AACS issues to the Mosaic Controller a *TurnEnd* event notification.
43. The Mosaic Controller directs the Camera to take an image.
44. The Camera exposes the CCD for a duration of time specified in the camera_exposure variable, and then issues a *ShutterClosed* event notification to the Mosaic Controller.
45. The Camera reads the CCD and if *image_compression* is turned on then applies the image compression algorithm to the raw data and stores it in a local buffer.
46. The Mosaic Controller directs the AACS to continue with the mosaic execution by writing to the AACS variable *picture_taken*.
47. These last 9 steps continue until all images have been taken.
48. When the last image has been taken, the Mosaic Controller issues a *MosaicEnd* event notification to the Client.

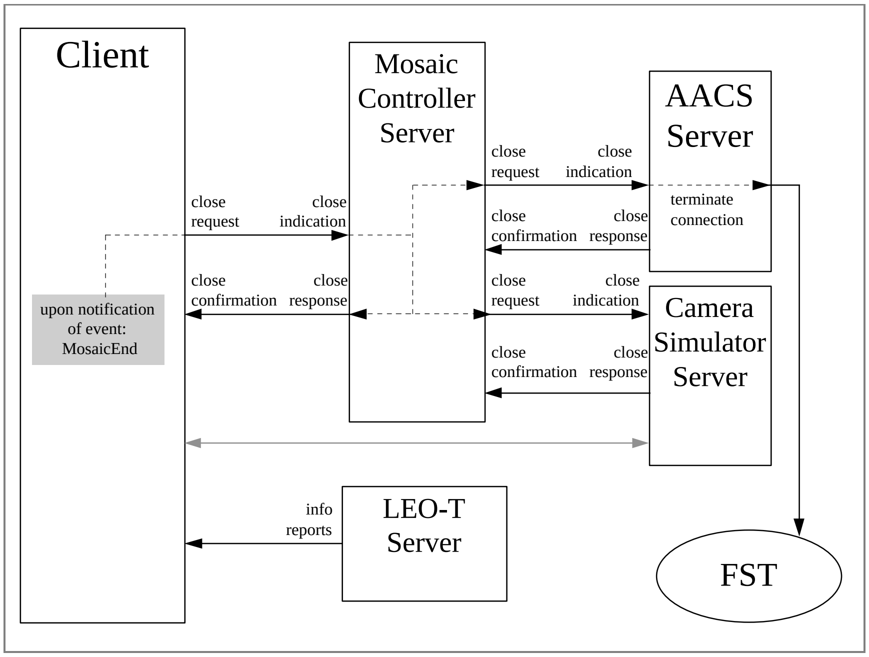

Figure 8 Termination of Connection On Board the Spacecraft

<u>Termination of connections on-board the spacecraft (see Figure 8):</u>

49. The Mosaic Controller concludes the connection to the Camera.
50. The Camera responds positively to the conclude message.
51. The Mosaic Controller concludes the connection to the AACS.
52. The AACS responds positively to the conclude message.
53. The AACS terminates the connection to the FST.

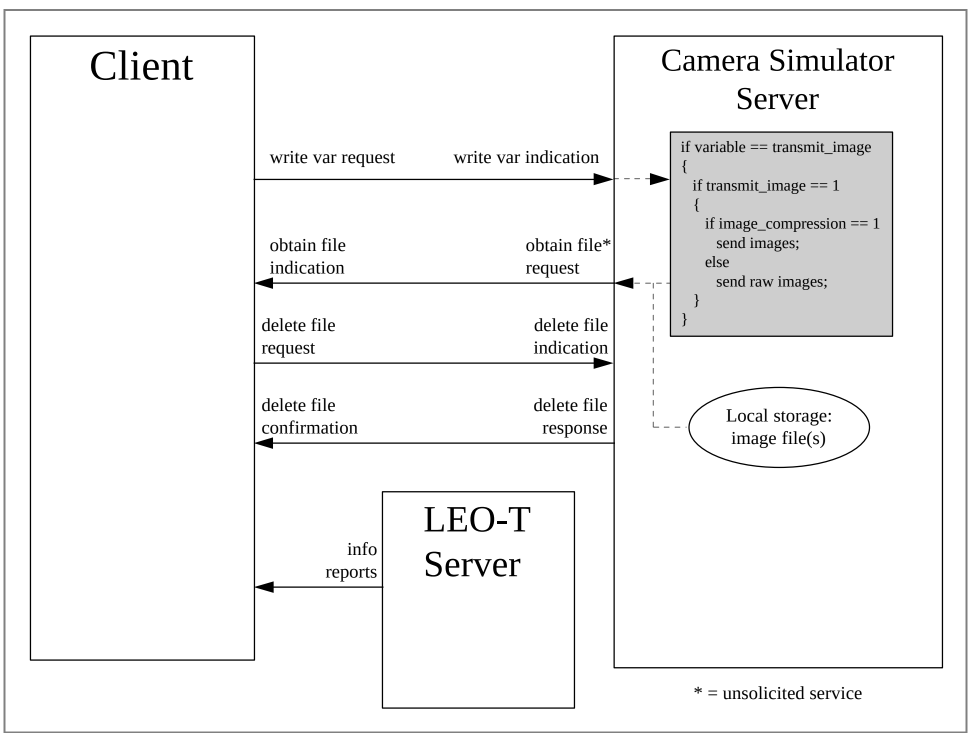

Figure 9 Image File Transfer from the Camera to the Client

Transmission of the image file by the spacecraft to the ground (see Figure 9):

54. The Client directs the Camera to return all images by writing to the Camera's *transmit_image* variable. The Camera issues a response and the Client receives the confirmation.
55. The Camera issues an obtain file indication to the Client (this is an unsolicited message, i.e., the Client did not issue an obtain file request first).
56. The Camera sends the image files to the Client using the MMS file transfer services.
57. The Client issues a delete file request to the Camera for each image file. For each request the Camera receives an indication, generates a response and the Client receives the confirmation.

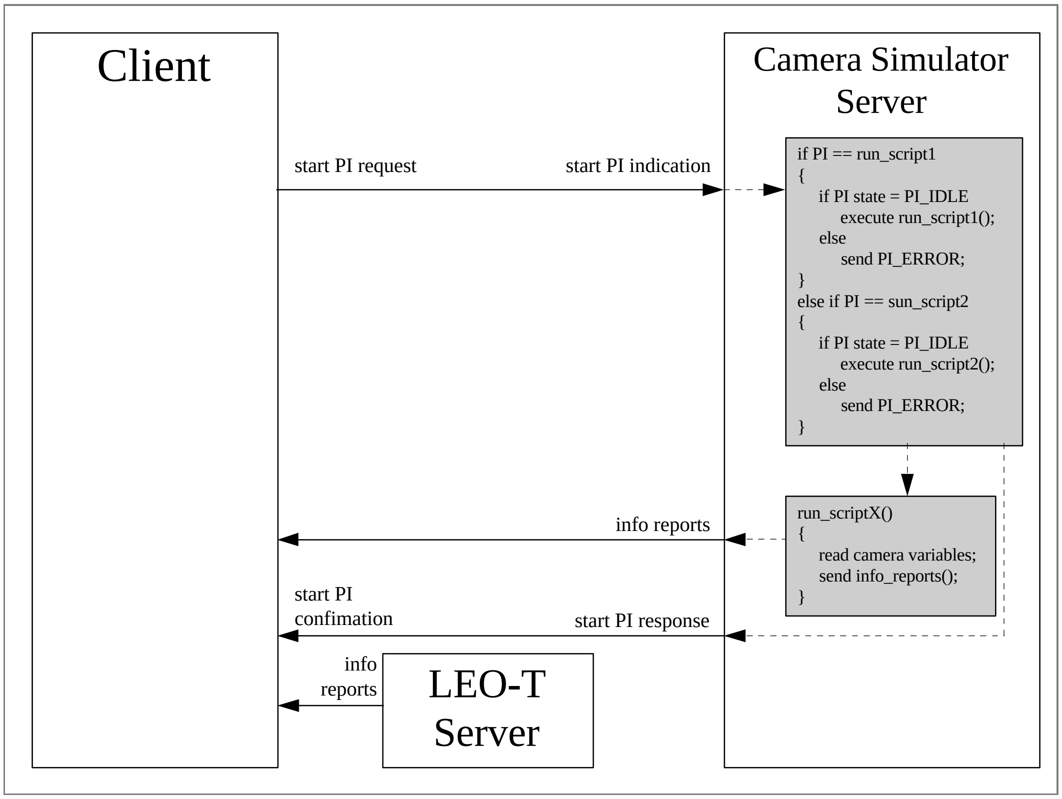

Figure 10 Maintenance Script Execution by the Camera Simulator

Execution of Camera Maintenance Scripts (see Figure 10):

58. The Client sends a start program invocation request. The Camera receives the indication, searches the name of the requested program invocation, and if found and if the program invocation state is IDLE then the program is executed.
59. The run_scriptX programs are implemented as simple scripts that are parsed by a function named do_maintenance() in the Camera. These scripts can be overwritten by the Client. The Camera issues information reports to the Client reporting the status of the maintenance script execution.
60. After the execution of the run_scriptX programs the Camera issues a program invocation start response and the Client receives a confirmation.

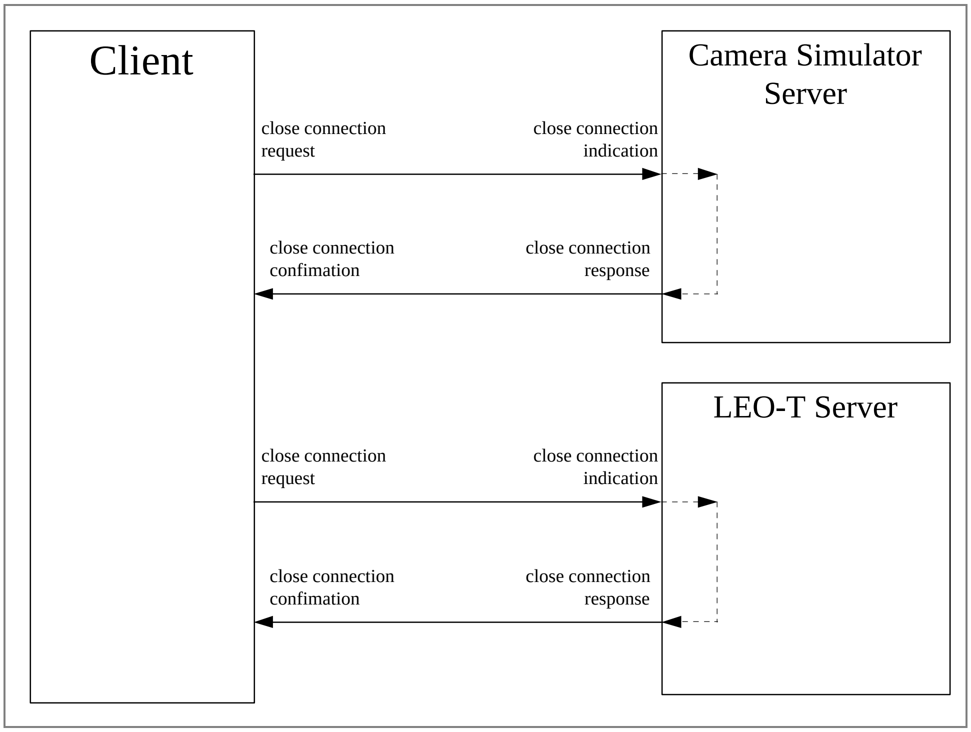

Figure 11 Termination of Camera and LEO-T Connections by the Client

Termination of connections by the Client (see Figure 11):

61. The Client concludes the connection to the Camera by sending it a *conclude* request.
62. The Camera responds positively to the *conclude* message.
63. The Client concludes the connection to the Ground Terminal.
64. The Ground Terminal responds positively to the *conclude* message.

Configuration

The mosaic scenario was implemented using the following configuration:

- Client, Mosaic Controller Server, Camera Simulator Server:
    - SUN Sparc Solaris 2.4 (smocc1)
    - MMS-EASE 133-015 V6.0.3
    - SunLink OSI 8.1
- Attitude Control Server:
    - SUN Sparc Solaris 2.4 (smocc1)
    - MMS-EASE 133-015 V6.0.3
    - SunLink OSI 8.1
    - Tramel 3.1 proxy (used to communicate with the FST)
- FST Spacecraft Flight Software (rtc0):
    - Heurikon HKV4F Board
        - M68040 Processor
    - VxWorks 5.2
    - Tramel 3.1 (FST based messaging system)
    - TCP/IP
- Ground Terminal (leo-t1)
    - SUN Sparc Solaris 2.4

- MMS-EASE 133-015 V6.0.3
- SunLink OSI 8.1

Below is a table of the 18 MMS services used to implement this scenario.

Table 5 MMS Services Used in the Mosaic Scenario

Categories of MMS Services	**Services Used**
Context Management Services	Initiate Conclude
VMD Support Services	Identify GetNameList
Variable Access and Management Services	Write Read InfoReport
File Access and Management Services	ObtainFile DeleteFile FileDirectory
Event and Alarm Management Services	Define Event Condition Define Event Enrollment Event Notification
Domain Management Services	Initiate Download Download Segment Terminate Download
Program Invocation Management Services	Create PI Start

Results

Only 21% of the services provided by MMS were used to implement this imaging mosaic scenario. The Operator Communication and Journal Management Services were not used. Although it seems like the Journal Management Services may provide capabilities that would be useful, for example, in downlinking engineering telemetry summaries to the ground. The File Access and Management Services were used to downlink the image files to the client, but it may be more efficient to use native file transfer services for this function.

From this rudimentary application of one commercial messaging system to a typical deep space mission science scenario, commercial messaging systems look promising for use in space mission environments and thus further study would be beneficial. The capabilities offered by messaging systems offer a comprehensive set of services needed by spacecraft intracommunications: variable access, program execution, shared resource management, and event management.

This scenario assumed several things that are not quite true in deep space missions. One major assumption is the “real-time” feel to the mosaic commanding and execution. In reality due to the long delay times, such a mosaic command would be uplinked ahead of time, stored on board the spacecraft, and executed at a later time. Thus the dialogue between the client and the mosaic controller for a variable write request, domain download, or event notification, indeed for any request that requires a confirmation would be untenable. The protocol for confirmed services would need to be modified to take into account long light time delays or even communication breaks that would exist in the ground to space link.

On board the spacecraft the issue of time critical control loops comes up. For tight control loops like those that exist in spacecraft attitude maintenance dialogues between devices, we have not shown that an MMS-like messaging system would be appropriate. Further work is planned to try to answer such questions as these.

This work was done at the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

References

[1] Systems Integration Specialists Company, Inc. "Overview and Introduction to the Manufacturing Message Specification (MMS)", Revision 1.0, June 1994.

[2] D. Gilchrist and L. Neitzel. "The Applicability and Compatibility of MMS (ISO 9506) to Space Station Return Link Path Service". Space Station Freedom Contract NAS 9-18200, Work Package-2. June 1989.

[3] W. Randy Heuser. "The Adaptation of Industrial Protocols for a Space Messaging Service". International Conference on Reducing the Cost of Spacecraft Ground Systems and Operations, Rutherford Appleton Laboratory, Oxfordshire, England, September 1995.